Report for Algorithms & Analysis Assignment 1

Cheng Chen s3728207

Bowen Zhang s3617571

We (I) certify that this is all our (my) own original work. If we (I) took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. We (I) will show we (I) agree to this honor code by typing "Yes": Yes.

Experimental Setup

-Data Scenarios

Scenario 1: Split Node - Ratio: 10

Bigger data set is running unstable on our machine, so we decided to run within the supplied data file which is 2000 lines of data. A ratio of 10 is sufficient to see the difference between the two implementations.

Scenario 2: Find Node - Ratio: 10

Bigger data set is running unstable on our machine, so we decided to run within the supplied data file which is 2000 lines of data. A ratio of 10 is sufficient to see the difference between the two implementations.

Scenario 3: Print Node - Ratio: 10

Bigger data set is running unstable on our machine, so we decided to run within the supplied data file which is 2000 lines of data. A ratio of 10 is sufficient to see the difference between the two implementations.

-Sizes of data:

20, 200 and 2000

We used these sizes because these sized allow us to see the performance difference between the two implementations and their trend towards bigger size of data.

-Generation of scenarios

We wrote code to generate data from the supplied data set for different scenarios.

-Timing

We did consecutive runs for each test to get the average time.

We did 10 for each test to generate the data.

We remove the quickest 2 and slowest 2 to eliminate error due to unstable system and calculate average using the remaining 6 runs. **Excluded** data are in **bold** font.

-Platform: Linux

Evaluation

Scenario 1: Split Node

Data Format in: 'Sequential Time' - 'LinkedList Time' (nanoseconds)

Test	20	200	2000
1	**69610**-**101661**	**800405**-**974585**	**15823317**-**39290974**
2	40343-**86936**	285281-**847699**	10794908-**38884734**
3	39139-76919	291013-405936	**10865149**-38783666
4	41737-52369	279448-396337	10113687-36493250
5	**109912**-36784	**263813**-374276	10094858-36678227
6	**35006**-**30265**	264762-399729	10144976-36320292
7	37004-31493	**261243**-**353851**	**8613246**-**27128901**
8	39162-38976	**309028**-409400	10139516-36155877
9	35820-**31338**	280598-363415	10103804-**35965566**
10	**29287**-34826	274492-**346801**	**10016664**-36119442
Avg.	38867-45227	279265-391515	10231958-36758459

We found that when input data gets larger, it is obvious that the sequential implementation takes less time than the LinkedList implementation.

From our expectation, LinkedList representation should be faster to find the target. But the result shows otherwise. The difference is that in LinkedList representation every time we split a node, it has to allocate memory for the new nodes. But in sequential representation the memory is already allocated, and it just need to assign values.

Scenario 2: Find Node

Data Format in: 'Sequential Time' - 'LinkedList Time'

Test	20	200	2000
1	**4317**-**8764**	11608-**7339**	**208441**-**99403**
2	2180-1998	13535-7311	123471-41192
3	**2234**-1944	**10927**-**2885**	123986-40753
4	2126-1784	**11495**-2959	122989-37628
5	2097-1699	13929-**2926**	**119194**-37832
6	2052-1688	19689-6494	124890-**46136**
7	2168-1748	**21532**-6036	**120872**-**27416**
8	2060-**2341**	19577-5220	123110-**37600**
9	**1593**-**1211**	21465-**8248**	**125177**-37639
10	**1586**-**1096**	**22302**-5830	123517-37622
Avg.	2113-1810	16633-5641	123660-38777

We found that when input data gets larger, it is obvious that the LinkedList implementation takes less time than the sequential implementation.

This is expected since there are empty spaces in sequential representation and when finding a node, the empty spaces are also visited and compared. In LinkedList representation there are no empty spaces, so the total visits and compares are less than that in sequential representation.

Scenario 3: Print Node

Data Format in: 'Sequential Time' - 'LinkedList Time'

Test	20	200	2000
1	**10681624**-**486208**	**256681**-**209648**	**2019239**-**2275106**
2	142185-**138418**	**177145**-156193	**773429**-**860924**
3	140248-132308	111726-102192	376518-412268
4	143631-127029	127103-109075	205465-261293

5	**102880**-103081	126270-102124	203802-271701
6	**102968**-98835	**103751**-98423	189246-**168864**
7	**161153**-137880	**103796**-**93656**	**152021**-**183730**
8	108971-**87827**	126923-**94679**	153663-189300
9	118996-**88742**	142639-**156375**	186440-187655
10	110204-100240	111231-95688	**151722**-202094
Avg.	127372-116562	124315-110615	219189-254051

We found that the difference between the two implementations are small enough to ignore under all input sizes.

This is expected since to print all the nodes, it has to go through all the nodes in the tree regardless of which representation is used. Leading to complexity class of O(n).

Recommendation

For Split Node:

We recommend using Sequential Representation. We can see that when input size gets large (2000), sequential uses significantly less time compared to LinkedList. And the difference ratio is larger than medium input size (200). We can see that when input size gets even larger, the difference will also become bigger which means sequential is much more efficient.

For Find Node:

We recommend using LinkedList Representation. When input size gets large, by looking at 200 and 2000, sequential uses about 3 times the time of LinkedList representation. When input size gets even larger, LinkedList representation should always be 3 times more efficient than sequential representation.

For Print Node:

Both perform identically so both can be used.

Under all input size we tested (20, 200, 2000), both representations have identical performance. We predict that when input size gets larger, they will still perform roughly the same.

Part C

C1.

	O(1)	O(log(n))	O(n)	O(n!)
Find parent	No	No	Yes	Yes
Find a node	No	No	Yes	Yes
Print all nodes	No	No	Yes	Yes

Find parent:

Basic operation: comparison

Input size: n - 1

C(N) =$\sum_{i=1}^{n-1} 1$ = n - 1 ∈ O(n)

Find a node:

Basic operation: Compare value

Input size: n

C(N) =$\sum_{i=1}^{n} 1$ = n ∈ O(n)

Print all nodes:

Basic operation: Comparison

Input size: n

C(N) =$\sum_{i=1}^{n} 4$ = 4n ∈ Ω(n)

n! is larger than n so they also belong to O(n!)

C2.

1. In the worst case, he needs to ask N questions.

   In the average case:

   $C_{avg =}$ 1*1/N + 2*1/N+…….+N*1/N

   =(1+2+3……+N)*1/N

   =(1+N)*N/2N

   =(1+N)/2

he needs to ask (1+N)/2 questions.

1 second each question:

worst case: 14 billion seconds which is 443.937 years

average case: (1+14 billion)/2 = 7000000000.5 seconds which is 221.969 years

2. In the worst case:

Assume the question is only "at most" and the answer is only yes, or no. assume the number of question is n. the worst case occurs when |n.value – (n – 1).value| == 1 and |n.value – (n – 2).value| == 1. The basic operation is comparison, aka question

C(N) = C(N/2) + 1   and   C(1) = 1

Since C(N) = C(N * 2 ^ -1) + 1 = C(N * 2 ^ -2) + 2;

Assume N = 2 ^ k;

$C(2^k) = C(2^{k-1}) + 1$ for k > 0;

$C(2^0) = 0$;

$C(2^k) = C(2^{k-1}) + 1$ substitute $C(2^{k-1}) = C(2^{k-2}) + 1$

$= C(2^{k-2}) + 2 = C(2^{k-3}) + 3 = \dots = C(2^{k-i}) + I = \dots = C(2^{k-k}) + k$

Therefore $C(2^k) = C(1) + k = k + 1$

since $n = 2^k$, $k = \log_2 n$, $C(N) = \log_2 N + 1$;

So in the worst case, he needs to ask ($\log_2 N + 1$) questions

In the average case:

$C_{avg} = 1*1/(\log_2 N + 1) + 2*1/(\log_2 N + 1) + \dots\dots + (\log_2 N + 1)*1/(\log_2 N + 1)$

$=(\log_2 N + 2)/2$

He needs to ask ($\log_2 N + 2$)/2 questions

If N = 14 billion:

Worst case: 34.7 which is 35 questions

Average case: 17.85 questions